PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 4 : G06F 9/46

A1

(11) International Publication Number: WO 90/0595

(43) International Publication Date: 31 May 1990 (31.05.90

(21) International Application Number: PCT/US88/04061

(22) International Filing Date: 14 November 1988 (14.11.88)

(71)(72) Applicant and Inventor: HUMMEL, Heinrich [DE/DE]; Agnes-Bernauer-Strasse 26, D-8000 München 21 (DE).

(81) Designated States: AT (European patent), BE (European patent), CH (European patent), DE (European patent), FR (European patent), GB (European patent), IT (European patent), LU (European patent), NL (European patent), SE (European patent), US.

Published
With international search report.

(54) Title: METHOD OF HANDLING UNINTENDED SOFTWARE INTERRUPT EXCEPTIONS

(57) Abstract

Unintended software-interrupt-exceptions that are detected by the INTEL-80286 ® or the INTEL-80386 ® microprocessors while operating in the Protected-Mode and being known as interrupts 0, 4, 5, 6, 8, 12, 13, 16, are handled by using exclusively task-gates, by modifying the task-state-segment of the interrupted task while the interrupt task runs to so that AFTER executing the interrupt return instruction at a system-central the user's task-specific exception handler can safely be called and by providing a method to exit the user-task-specific exception-handler for resuming the normal operation at an appropriate program point which the task has already passed before the exception occurred, which essentially consists of saving and retrieving all register values and stack data as were actual when the task passed that point in the regular processing. The same principle applies to other processors with exception detection capability. A futural processor is conceived that provides instructions for assigning use-task-specific exception handlers, creating and jumping to recovery points.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	ES	Spain	MG	Madagascar
AU	Australia	FI	Finland	ML	Mali
BB	Barbados	FR	France	MR	Mauritania
BE	Belgium	GA	Gabon	MW	Malawi
BF	Burkina Fasso	GB	United Kingdom	NL	Netherlands
BG	Bulgaria	HU	Hungary	NO	Norway
BJ	Benin	IT	Italy	RO	Romania
BR	Brazil	JP	Japan	SD	Sudan
CA	Canada	KP	Democratic People's Republic of Korea	SE	Sweden
CF	Central African Republic			SN	Senegal
CG	Congo	KR	Republic of Korea	SU	Soviet Union
CH	Switzerland	LI	Liechtenstein	TD	Chad
CM	Cameroon	LK	Sri Lanka	TG	Togo
DE	Germany, Federal Republic of	LU	Luxembourg	US	United States of America
DK	Denmark	MC	Monaco		

1.Description:

===============

Method of handling unintended software interrupt exceptions:

The invention refers to a method of handling software interrupt exceptions which are not caused on purpose but occur due to errorprone programs that are particularly processed on the INTEL-80286 (TM) or INTEL-80386 (TM) processor in case these processors operate in the so called PROTECTED MODE, whereby its basic principle can also be applied to the exception handling of unintended (software interrupt) exceptions of any other existing exception-detecting processor e.g. like MOTOROLA's 68020 (TM) and whereby the method may even give the incentive for creating enhanced hardware processors that do support this method as well.

The method is described in detail based on the INTEL 80286 (TM) processor.

Using the INTEL 80286 (TM) processor the utilization of this invention may be appropriate for exceptions as may be due to

- Division by zero (known as interrupt 0),
- INTO -detected overflow (known as interrupt 4),
- exceeding a boundary range (known as interrupt 5),
- invalid Operation code (known as interrupt 6),
- double faults (known as interrupt 8),
- stack errors (known as interrupt 12),
- all kind of errors that can be categorized as general protection errors (known as interrupt 13),
- processor extension errors (known as interrupt 16),

which are by nature exceptions that are not intended on purpose.

Based on the consideration that these exceptions are not errors by their own but rather report errors that have been made at some other places by wrong or missing high-level-programming-language statements this approach of handling these exceptions will not try to restart the faulty instruction but to provide full opportunity to the user task to investigate and deal with the current situation in the logical level of the used high-level-programming-language and to provide the capability to resume the normal operation thereafter, no matter which of the listed exception occured, yes, even if the complete stack has been destroyed prior being reported by a stack error exception (interrupt 12).

As is standard this invention pursues to call a user task specific exception handler (see literature "INTEL 286 Operating System Writer's Guide" page 6-7 to 6-9) however claiming novelty for the way to do so highlighted by the execution of the interrupt return instruction before the user task specific exception handler has even been called.

This is done to prevent that further exceptions that might be caused by the user task specific exception handler can accrue to double faults or even to processor shutdown.
Furthermore this is done in a way so that the user task specific exception handler can safely be reached and started, i.e. that no leftover data due to the occured exception may cause additional exceptions nor that data is left behind that might be troublesome for the next exception.

Dictated by the worst cases where an interrupt task gate is definitely required, the solution assumes to use an interrupt task gate in the interrupt descriptor table for all above listed interrupt numbers. The task gate may refer to a task descriptor in the global descriptor table, which will refer to a task state segment, which will refer to an interrupt task program.

This interrupt task program may save all data about the occurance of the exception, like the total stack segment of the interrupted task as well as the contents of its task state segment (which means essentially all register values), for a later failure report - which is reasonable but not significant for this invention -, then using the ALIAS-descriptor technique modify the task state segment of the interrupted task, particularly

- the fields for CS and IP so that by executing the interrupt return instruction the interrupted task would continue at a system central address, let's call it Z,

- e.g. the fields for BX and CX so that by executing the interrupt return instruction the registers BX and CX would contain the address of the user task specific exception handler (there are wellknown ways how to assign/where to store/ from where to get the address of the user task specific exception handler),

- the fields for DS and ES to any valid and present datasegment's selector so that by executing the interrupt return instruction no task switch error (known as interrupt 10) may be caused by these field entries,

- the field for the dynamic SS entry by copying from the proper static SS entry with regard to the appropriate privilege level 0 to 2, if this can be done unambiguously, - otherwise we may trust it and keep it as is, as the user program normally doesn't touch that value,

- the fields for SP and BP to the limit value minus 1 of the interrupted task's stacksegment so that after the return to the interrupted task new data can safely be put into the stack, e.g. in order to execute the calling of the user task specific exception handler,

- e.g. the fields for DI and SI to the address of the segment where we might have saved all above mentioned data concerning the occurance of the exception.

The interrupt task program must also clear the TASK_SWITCHED bit in the machine status word. The interrupt task program or programs that is or that are reached via the task gates for exception 8, 12, 13 must also pop the error code from the stack of the interrupt task prior switching back to the previously interrupted task.

By executing the interrupt return instruction the previously interrupted task will continue at the system central address (Z) so that from there on new values may be safely entered into and read from the stack, while all data that had been written into the stack prior to the occurance of the exception will be lost and needs to be recovered by an unorthodox method - which is an essential part of this invention as well.

At the system central address Z the user task specific exception handler is to be called (its address may be stored in the registers BX and CX), eventually provided with the input (parameter) of the address of the datasegment that contains the data about the occurance of the exception, unless the user task hasn't been assigned the address of a specific exception handler. In the latter case a system master exception handler program may be called, by using a task gate, to provide more general reaction service.

The user task specific exception handler is supposed to investigate the current situation and deal with the problem appropriately, to do this in the adequate logical level (normally in the used high level programming language), then to determine the best suitable point of the task's program, let's call it a recovery point, and to "jump" there, not just by setting code segment register (CS) and instruction pointer register (IP) to that point's address but by loading all stack data and all register values (inclusively CS and IP at last) as has been actual when the task's program flow passed that point the last time. This kind of "jump" must have been supported by the user's program when it passed that program address in the regular processing by writing all the data, i.e. stack and all register values enhanced by the self-descriptive information of how many bytes are saved, into a particular datasegment of appropriate size whose address must be saved at a distinctive memory address for being retrieved again.

As theoretically the same program code may be processed while using different tasks, the "jump" to the recovery point may only, be allowed if the actual stack segment value is equal to the stack segment value as retrieved from that datasegment. Legal "jumps" to the same recovery point while being in different tasks must be matched by using different, task specific datasegments for storing the recovery point's characterizing data.

In case the user task specific exception handler does not try to exit by "jumping" to a recovery point or in case the attempted "jump" to a recovery point is rejected it will terminate by the normal subroutine-return-instruction which allows us to call the system master exception handler by using a task gate which in any particular program branch may either exit by "jumping" to any other recovery point or initiate the rebooting of the system.

Using this method a computer system or an executed user program cannot crash anymore in all applicable situations. Multi-user-, multi-application, multitask-systems which otherwise might even deteriorate each other and wind up in softrestarting/hardrestarting the entire system will benefit as each task will be able to stay in full control by its own. Without this method the exception detection capability may be considered as being disadvantageous, especially if less damage is caused by processing undefined data rather than by crashing the system; with it the user may taylor his/her modules to get the most out of the exception detection capability.

The implementation of this invention is by its nature an ideal enhancement of any Operating System that has to reflect hardware specifics as is the detection of exceptions. Part of it can be categorized as system configuration, part of it as Operating System service routines that are to be provided to the user, e.g. to "define a recovery point", which essentially means to save the actual register and stack values in a retrievable way, or to "jump to a recovery point", which means to retrieve and load these values into registers and stack, or to initialize involved data.

In case it is applied "outside" of any standard Operating System it may improve just those systems/devices (e.g. PC's) of those manufacturers that provide it in addition to/and overruling the Operating System, or in case it is applied "outside of the base system" but being a part of certain application software packages these may benefit e.g. by coping with internal errors or inproper data input by their customers.

2.Patent claims:
================

1. Method to cope with the problems resulting from software interrupt exceptions, which occur due to program errors rather being intended on purpose and which are detected by the INTEL-80286 (TM) or the INTEL-80386 (TM) processor while operating in PROTECTED MODE, by enabling the calling of a user task specific exception handler upon the occurance of exceptions as may be given by software interrupt numbers 0,4,5,6,8,12,13,16

whereby the method is characterized particularly, by using task gates for all respective entries in the interrupt descriptor table, i.e. for entries 0,4,5,6,8,12,13,16, which may refer to task descriptors in the global descriptor table which may refer to task state segments which may refer to interrupt task programs,
and by the activities of these interrupt task programs, which must pop the ERROR CODE however only in case the interrupt task programs are determined for the interrupts 8,12 or 13, while all of them must clear the TASK_SWITCHED bit of the machine status word and must modify the task state segment of the interrupted task so

that by executing the interrupt-return instruction a task switch back to the interrupted task will occur in such a way that this task will continue to run at a system central address, that then the address of the user task specific exception handler is available in a pair of suitable registers, that then new data can safely be written into and read from the stack, i.e. that then the stackpointer and basepointer are set to the limit minus 1 of the actual stacksegment, that the data-segment-register and the extra-segment-register cannot cause a task switch error (interrupt 10) being explicitly set to any valid and present datasegment,
and by then returning to the previously interrupted task, which means by continuing at a system central address,
and by then calling the user task specific exception handler whose address is available in a suitable pair of registers so that the individual application-dependent user task specific exception handler, which may as well cause unintendend exceptions, may not wind up in double fault or processor shutdown,
and by the technique how to exit the user task specific exception handler which is done by "jumping" to a program point of the task, which I like to call recovery point and which is appropriate to resume the normal operation after having ingestigated and handled the current problem

situation from the user application's point of view, that is done by retrieving and loading all register values and stack data as have been actual when the regular program processing passed the address of that recovery point at the last time prior to the occurance of the exception, and by the technique how to store these register values and stack data when the task program passes a program point that is suitable for being retrieved upon the occurance of a software interrupt exception in order to resume the normal operation as is done by storing register values and stack data into a datasegment of appropriate size in a retrievable way i.e. together with additional self-descriptive information as the number of stack words is a variable information and by memorizing the address of the datasegment at a well defined place.

2. Method to cope with the problems resulting from software interrupt exceptions, which occur due to program errors rather being intended on purpose and which are detected by any microprocessor that provides the capability to get to a separate program area upon the occurance of such exceptions and that provides as well an interrupt-return instruction for counter-processing what has been processed due to the exception detection

which is characterized

by taking preparations prior to the execution of the interrupt-return instruction in such a way,

that in the moment of executing the interrupt-return instruction all processor knowledge and processor impact about the occured exception is cleared and no further exception can be caused due to register values as has been actual for the faulting instruction, whereby registers and other data may freely be modified to achieve this,

that after the execution of the interrupt-return instruction the program will continue at a certain system central address and not at the address of the faulting instruction,

that then appropriate registers are set so that the complete stack is made available,

if necessary, that then the address of the user's exception handler is available,

by then executing the interrupt-return instruction,

by then calling the user's exception handler which is supposed to investigate and handle the actual problem situation from the application dependent point of view and to determine the most suitable program point for

resuming the normal operation,
by enabling the user's exception handler to "jump" there which is done by retrieving all register values and stack data as has been actual when the regular program flow passed that program point the last time prior to the occurance of the exception,
by supporting such "jumps" which is done by saving all actual register values and stack data in a retrievable way at moments when the regular program processing passes program points that are suitable for being jumped-to for resuming the normal operation.

3. Enhancement of any microprocessor that is able to detect exceptions which are defined according to its own specific design and which typically occur due to the execution of errorprone programs rather being used on purpose
which is characterized by

providing an extended set of assembler code instructions, particularly an assembler code instruction for assigning a task the address of a procedure to be determined as the task's exception handler program,
particularly an assembler code instruction for creating a

recovery point, i.e. for saving all register values and stack data as are valid when this instruction is invoked, whereby any kind of token is returned by an output parameter of any type, which can be used by another instruction to retrieve the hereby saved register values and stack data and which is mentioned next,
particularly an assembler code instruction to jump to a recovery point, i.e. to retrieve register values and stack data due to an input parameter which contains the token that has been returned by the instruction just mentioned before,
and if necessary an assembler code instruction to release a recovery point, i.e. to release the required memory where stack data and register values of a program point are stored in case the overall design requires it to manage limitation aspects,

so that the user has only to call the new instructions at the appropriate places, e.g.
at the beginning of the task the one to assign a user-task-specific exception handler,
at an appropriate program point for the resuming of the normal operation of the task after the occurance of an exception the one to create a recovery point,
at the end of any branch of the user-task-specific

exception handLer the one to jump to a recovery point.

INTERNATIONAL SEARCH REPORT

International Application No. PCT/US88/04061

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) 6

According to International Patent Classification (IPC) or to both National Classification and IPC

IPC(4): G06F 9/46

US. CL.: 364/200

II. FIELDS SEARCHED

Minimum Documentation Searched 7

Classification System	Classification Symbols
U.S.	364/200, 300, 900

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched 8

III. DOCUMENTS CONSIDERED TO BE RELEVANT 9

Category *	Citation of Document, 11 with indication, where appropriate, of the relevant passages 12	Relevant to Claim No. 13
A	US, A, 4,205,370 (HIRTLE) 27 MAY 1980, SEE ABSTRACT	1,2,3
A	US, A, 4,535,456 (BAUER et al.) 13 AUGUST 1985, SEE ABSTRACT	1,2,3,

* Special categories of cited documents: 10

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search 24 MAY 1989	Date of Mailing of this International Search Report 13 JUL 1989
International Searching Authority ISA/US	Signature of Authorized Officer R.B. Zache R.B. ZACHE